A Mash-up Architecture for Learning Environments and Knowledge Management Systems

Bernhard Hoisl

Institute for Information Systems and New Media
Vienna University of Economics and Business (WU Vienna)
UZA II, Augasse 2-6, 1090 Vienna, Austria
bernhard.hoisl@wu.ac.at

Abstract: In recent time a new trend can be recognised on the Internet in general and especially in learning environments and knowledge management systems by moving away from monolithic 'one-provider-fits-all' to a combinatorial 'mixing-pieces-together' approach. Mashing-up stands for the re-use, -combination, and -organisation of small software artefacts of clearly defined functionality. Subsequently, mashed-up learning or knowledge management systems describe the idea of highly customisable environments shifting substantial personalisation possibilities from administrators to the end-users. This shift has significant impacts on infrastructure requirements of learning and knowledge management systems as well as on software design decisions. Therefore, this paper describes an approach for a mash-up architecture, making use of small software artefacts which are capable of being easily integrated in various systems.

1 Introduction

Mashed-up Learning Environments (LE) or Knowledge Management Systems (KMS) have the great advantage of being highly adaptable, customisable, re-arrangeable, -combinable, -usable, and interoperable [WMS08]. By mashing-up (pre-existing) software artefacts, an advantage of creating new and/or combined functionalities can be gained. As a practical example we will explain our architecture developed in the 'Language Technologies for Lifelong Learning' (LTfLL)[1] project for designing a software system closely following a mash-up approach (Section 2: Architecture). LTfLL's consortium partners are developing different kinds of software artefacts with the help of multiple and varying technologies (Section 3: Application Logic). Therefore, the chosen integration approach must allow for combining these artefacts with a high degree of individual freedom in software design choices. With the mash-up approach project partners can develop software in their own context and plug-in their artefacts in an integrative environment, thus generating a set of customisable services (Section 4: Widgets). On the one hand, the advantage of this approach from the learner's or knowledge worker's point of view is that heterogeneous software systems are plugged into a single environment: they can be arranged individually but feel and look like one coherent software system. On the other hand, benefits for software developers and system administrators are that a modularised system like this can easily be transferred to

[1]For more information visit http:/www.ltfll-project.org.

different platforms with less effort, making it highly interoperable and re-usable (Section 5: LE and KMS Integration). One challenge of mashing-up software artefacts is the need for developing well-defined interfaces, e.g. for shared data interchange. Providing services with standardised interfaces brings the question of an integration strategy to a higher level, eclipsing technological decisions of programming languages or database management systems (Section 6: Discussion).

2 Architectural Overview

As can be seen in Figure 1 the system design is following a classical three-tier server-client architecture with its data, application, and presentation tier. Furthermore, the proposal makes use of an additional middleware layer connecting the application logic and the Graphical User Interface (GUI). For the integration of the individual services a widget based approach was chosen. Widgets are encapsulated "client-side applications that are authored using Web standards and packaged for distribution. They […] can also be embedded into Web pages and run in a Web browser" [W3C10]. These widgets can be integrated in nearly every LE or KMS.

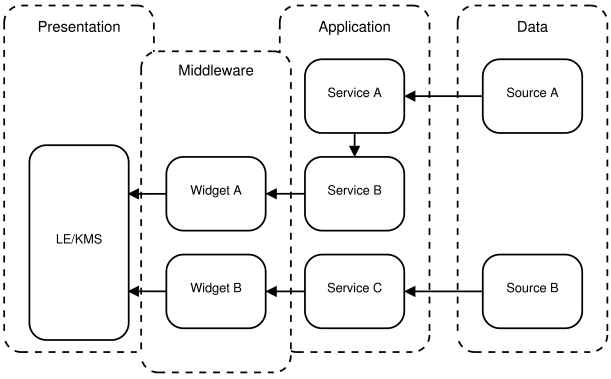

Figure 1: Four-tier architecture

The biggest architectural difference compared to traditional monolithic LE or KMS is that all components providing functionality are completely detached from the underlying software system. This having the advantage of being a very flexible solution and is a benefit in case of software re-usability.

3 Application Logic: Web-Services

The principles of our design approach is to develop a light-weight, easy to handle, multi-functional architecture not existent so far. Therefore, the primary goal is to build a scalable solution for independent deployments of software components with standardised interfaces. In-line with these requirements we have chosen to develop web-services communicating in a REST-styled (Representational State Transfer) way. Data

exchange between provided web-services and the presentation layer is done using structured message formats like XML (eXtensible Markup Language) or JSON (JavaScript Object Notation). Responses from web-services are used as input to other web-services or rendered in the user's web-browser by using widgets. These widgets are integrated in a LE or KMS creating the mash-up approach.

4 Middleware: Wookie Widgets

Widgets act as GUI and integration tool-kit. Management of widgets is done by using a dedicated widget engine called Apache Wookie [Wi10]. Wookie was chosen for several reasons: it has a large educational community, is open-source, is an Apache Incubator project, is standard compliant with the W3C widget working draft [W3C10], and has plug-ins already available for different LE, such as Wordpress, Moodle, and Elgg.

Widgets are handled and provided by the Wookie engine, whereas integration in LE or KMS is done by using plug-ins written in code native to the application. The integrating system is interacting with the widget through the plug-in which itself is using the REST API (Application Programming Interface) of Wookie for communication. A web-application that has widgets in it is called widget container [Wi08].

Widgets are created using web-standards, i.e. everything a 'normal' web-page could consist of. Widgets are executed on the client side, thus only scripting languages runable in a client's web-browser are allowed (e.g. JavaScript). Interactivity is modelled by calling web-services using AJAX (Asynchronous JavaScript and XML). Furthermore, as per W3C recommendation, widgets must obey to a particular file structure and must be packaged as ZIP archives before being uploaded to the Wookie engine.

Once available in Wookie all widgets are capable of built-in run-time functionalities like storing and retrieving settings, managing data shared among different widgets, trigger events between widgets, and calling external web-services. These functionalities are most important, because only then widgets are enabled to share data, invoke events, communicate to each other, and model basic workflows – pre-conditions for generating highly flexible LEs and KMSs.

5 Integration: LE and KMS

The integration of widgets is exemplified by using the open source social networking platform Elgg (http://www.elgg.org) as widget container application. Elgg handles user management, access control, community networking features and so on. Therefore, Wookie makes use of these already existent functionalities through the plug-in and delegates them completely to the widget container.

In the scope of the LTfLL project, a Wookie plug-in for Elgg was developed handling the integration of widgets [Ho10]. A user can display and arrange widgets through the use of Elgg's dashboard (Figure 2). As widgets are designed for small applications

placed side-by-side, their display dimensions are usually quite small [Go10]. For displaying larger content, widgets are resizeable by decoupling them from the dashboard. Users can rearrange their widgets or add/delete widgets on their dashboard. Another Elgg plug-in allowing for tabbed dashboards [Ka10] is extending the functionality for grouping widgets.

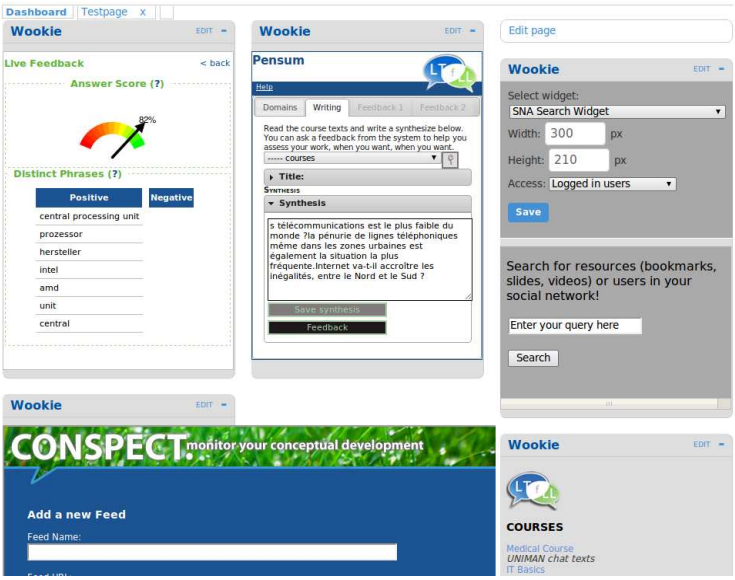

Figure 2: Wookie widgets integrated in Elgg

Within the LTfLL project Wookie's functionalities were also improved by implementing Inter-Widget Communication (IWC) components [HDW10]. Thus allowing widgets of one user to interact with each other. Therefore, data can be shared across different widgets and all widgets in one user space are notified of triggered events. For instance, by choosing a learning course from Widget A, Widget B automatically displays learning materials associated with this course and Widget C shows user's progress.

6 Discussion and Future Work

In this paper a prototype for a novel approach of developing mash-up LE and KMS has been introduced. A four-tiered and loosely-coupled system was explained gaining power from its high flexibility: on the one hand for learners and knowledge workers by re-arranging and re-combining components and on the other hand for developers by re-using software artefacts. Further benefits consist of high scalability of the proposed solution in conjunction with the possibility to independently deploy software components and being able to integrate them in a single or in multiple widget container applications.

Mashing-up small encapsulated software artefacts means each of them must have self-contained functionalities. As this approach has many advantages over traditional

architectures of LE and KMS, it limits itself through the increasing difficulty of software artefacts involved. It is easier to develop software exclusively for one platform than to claim that one solution will work for any environment. Therefore, software developers have to carefully think before developing mash-up LE or KMS if benefits of re-usability and interoperability outperforms the lock-in effect on one environment.

Short-time future work will head in the direction of adding new features to the Elgg plug-in, i.e. implementing the full set of functionalities provided by Wookie's RESTful API and developing an alternative view of choosing widgets in a gallery type of style. Furthermore, it is planed to extend the Wookie engine by improving IWC methods and possibilities and by implementing a token based authorisation process for web-services. As our approach is currently evaluated in real world settings, long-time future work will be based on the outcome of the evaluation. It is planned to revise our approach to optimise it to meet changing needs of the actual stakeholders: end-users of the system (learners and knowledge workers).

Acknowledgements

This work has been financially supported by the European Commission under the ICT objective of the 7th Framework Programme in the project 'Language Technologies for Lifelong Learning' (LTfLL).

References

[Di09] Dietl, R. et al.: Deliverable 2.2: Existing Services - Integrated. Heerlen: Open University of the Netherlands, 2009.

[Go10] Google: Gadget Guidelines. Available at: http://www.google.com/webmasters/gadgets/guidelines.html, accessed on 2010-10-24.

[HDW10] Hoisl, B.; Drachsler, H.; Waglechner, C.: User-tailored Inter-Widget Communication – Extending the Shared Data Interface for the Apache Wookie Engine. In: Proceedings of the 13th International Conference on Interactive Computer Aided Learning (ICL 2010). Kassel University Press, Kassel, 2010; pp. 1123-1131.

[Ho10] Hoisl, B.: Elgg Plugins – Wookie Widgets. Available at: http://community.elgg.org/pg/plugins/hoisl/read/385029/wookie-widgets, accessed on 2010-10-24.

[Ka10] Kanan, S.: Elgg Plugins – Tabbed Dashboard and/or Profile. Available at: http://community.elgg.org/pg/plugins/sammykanan/read/384603/tabbed-dashboard-andor-profile, accessed on 2010-10-24.

[W3C10] W3C: Widget Packaging and Configuration – W3C Working Draft 5 October 2010. Available at: http://www.w3.org/TR/widgets/, accessed on 2010-10-24.

[Wi08] Wilson, S.: Wookie – Widget Developer's Guide. Available at: http://trac.cetis.ac.uk/trac.cgi/wookie/attachment/ticket/21/widget_dev_guide.doc?format=raw, 2008, accessed on 2010-10-24.

[Wi10] Wilson, S. et al.: Welcome to Apache Wookie (Incubating). Available at: http://incubator.apache.org/wookie/, accessed on 2010-10-24.

[WMS08] Wild, F.; Mödritscher, F.; Sigurdarson, S.E.: Designing for Change: Mash-Up Personal Learning Environments. In: eLearning Papers, 2008(9), ISSN: 1887-1542.